USB HOST CONTROLLER WITH MEMORY FOR TRANSFER DESCRIPTORS

This invention relates to a bus system, and in particular to a bus controller, and to a device incorporating the bus controller.

More particularly, the invention relates to an integrated circuit which can be used as a host controller within an electronic device, in order to improve the efficiency of operation of the device.

In a conventional electronic device, operating as a USB host, the processor is able to write data into a system memory. A host controller integrated circuit is then able to read the data directly from the system memory. In order to be able to do this, the host controller needs to master the system memory. However, since the system memory is shared between the host controller integrated circuit and the system processor, this requirement that the host controller be able to master the system memory requires the use of a bus master, which is specific to the system processor. Moreover, while the host controller is mastering the system memory, the core function of the device, running under the control of the system processor, may be disrupted.

According to an aspect of the present invention, a host controller integrated circuit is unable to master the system memory, but instead acts purely as a slave. The embedded processor is then adapted to write the data to the host controller integrated circuit in the form of transfer-based transactions.

The invention will be described with reference to the accompanying drawings in which:

Fig. 1 is a block schematic diagram of a USB host in accordance with an aspect of the present invention.

Fig. 2 is a block schematic diagram of a host controller in accordance with another aspect of the invention.

Fig. 3 is a block schematic diagram of an alternative form of host controller in accordance with an aspect of the invention.

Fig. 4 illustrates the structure of the memory in the host controller of Fig. 2 or Fig. 3.

Fig. 5 is an illustration showing the format of software in the device of Fig. 1.

Fig. 6 illustrates the format of data written from the host microprocessor to the host controller.

Fig. 7 shows the structure of a transfer descriptor header, with which data is transferred.

Fig. 8 is a schematic representation of data to be transmitted, stored in the memory of Fig. 4.

Fig. 9 illustrates a method by which the data of Fig. 8 may be transmitted.

Fig. 1 is a block schematic diagram of the relevant parts of an electronic device 10, operating as a USB host. The invention is particularly applicable to devices such as mobile phones, or PDAs, in which the functional limitations of the microprocessor and the system memory are more relevant, rather than in personal computers (PCs). However, the invention is applicable to any device which can operate as a USB host.

It will be apparent that the device 10 will have many features, which are not shown in Fig. 1, since they are not relevant to an understanding of the present invention.

The device 10 has a host microprocessor 20, which includes a processor core 22, connected by a standard system bus 23 to a LCD controller 24, a DMA master 25, and a memory controller 26. The memory controller 26 is connected to a system memory 30 by means of a peripheral bus 32.

A host controller 40 is also connected to the host microprocessor 20 and the system memory 30, by means of the peripheral bus, or memory bus, 32. The host controller 40 has an interface for a USB bus 42, through which it can be connected to multiple USB devices. In this illustrated embodiment, the host controller 40 is a USB 2.0 host controller.

As is conventional, the host controller 40 is adapted to retrieve data which is prepared by the processor 20 in a suitable format, and to transmit the data over the bus interface. In USB communications, there are two categories of data transfer, namely asynchronous transfer and periodic transfer. Control and bulk data are transmitted using asynchronous transfer, and ISO and interrupt data are transmitted using periodic transfer. A Queue Transaction Descriptor (qTD) data structure is used for asynchronous transfer, and an Isochronous Transaction Descriptor (iTD) data structure is used for periodic transfer.

The processor 20 prepares the data in the appropriate structure, and stores it in the system memory 30, and the host controller 40 must then retrieve the data from the system memory 30.

Fig. 2 shows in more detail the structure of the embedded USB host controller 40.

As mentioned above, the host controller 40 has a connection for the memory bus 32, which is connected to an interface 44, containing a Memory Mapped Input/Output, a Memory Management Unit, and a Slave DMA Controller. The interface 44 also has a connection 46 for control and interrupt signals, and registers 48 which support the RAM structure and the operational registers of the host controller 40.

The interface 44 is connected to the on-chip RAM 50 of the host controller, which in this preferred embodiment is a dual port RAM, as will be described in more detail below. The memory 50 is connected to the host controller logic unit 52, which also contains an interface for the USB bus 42. Control signals can be sent from the registers 48 to the logic unit 52 on an internal bus 54.

As mentioned above, the on-chip memory 50 in this case is a dual port RAM, allowing data to be written to and read from the memory simultaneously.

Fig. 3 shows an alternative embodiment of the invention, in which common reference numerals indicate the same features as in Fig. 2. In this case, the on-chip memory 56 is a single port RAM, and data written to and read from the memory 56 is transferred through an arbiter 58, which again allows for effectively simultaneous access to the memory 56.

Fig. 4 shows the structure of the on-chip memory. As far as the structure shown in Fig. 4 is concerned, this is the same whether the on-chip memory is the dual port RAM 50 shown in Fig. 2, or the single port RAM 56 shown in Fig. 3.

As shown in Fig. 4, the RAM is effectively divided into two parts, namely a first part 70 which contains header and status information for the stored transfer descriptors TD1, TD2, ..., TDn, and which is itself subdivided into a portion 72 relating to asynchronous (bulk) transfers and a portion 74 relating to periodic (isochronous and interrupt) transfers, and a second part 76, which contains the payload data for those stored transfer descriptors TD1, TD2, ..., TDn.

This structure of the RAM has the advantage that the host microprocessor 20 can easily write and read all of the transfer descriptor headers together. This structure also makes it easy for the headers relating to periodic transfers to be scanned only once in each

micro-frame, while headers relating to asynchronous transfers are scanned continuously throughout the micro-frame.

This means that the time between transactions will be small and, equally importantly, it will be consistent from one transaction to another.

Fig. 5 is a schematic diagram showing in part the software operating on the host controller 40, in order to illustrate the method of operation of the device according to the invention.

The host controller 40 runs USB driver software 80 and USB Enhanced Host Controller Interface software 82, which are generally conventional.

However, in accordance with the present invention, the host controller 40 also runs USB EHCI interface software 84, which prepares a list of transfer-based transfer descriptors for every endpoint to which data is to be transmitted.

The EHCI interface software 84 is written such that it uses the parameters which are generated by the EHCI host stack 82 for the existing periodic and asynchronous headers, and can be used for all different forms of USB transfer, in particular high speed USB transfer, such as high speed isochronous, bulk, interrupt and control and start/stop split transactions.

The host microprocessor 20 writes the transfer-based transfer descriptors into the RAM 50 or 58 of the host controller 40 through the peripheral bus 32, without the host controller 40 requiring to master the bus 32. In other words, the host controller 40 acts only as a slave. The transfer-based transfer descriptors can then be memory-mapped into the RAM 50 or 58 of the host controller 40.

Advantageously, the built-in memory 50 or 58 of the host controller 40 is mapped in the host microprocessor 20, improving the ease with which transactions can be scheduled from the host microprocessor 20.

Moreover, as described above, the use of a dual-port RAM 50, or a single-port RAM 56 plus an arbiter 58, means that, while one transfer-based transfer descriptor is being executed by the host controller 40, the host microprocessor 20 can be writing data into another block space.

Fig. 6 illustrates the format of one USB frame, divided into multiple micro-frames, in which data is transmitted from the host controller 40 over the USB bus 42. As is conventional, multiple transactions, including transactions of different transfer types, may be sent within one micro-frame. Again, as is conventional, high speed isochronous transfer is always first, followed by high speed interrupt transfer, and full speed and low speed Start

Split and Complete Split transfers, with high speed bulk data occupying the remaining time in the micro-frame.

The transfer-based protocol allows the host microprocessor 20 to write a 1ms frame of data into the RAM 50 or 58 of the host controller (provided that the RAM is large enough to hold this data), such that this can be transmitted over the USB bus 42 without further intervention from the host microprocessor.

Fig. 7 illustrates the transfer-based protocol for supporting high-speed USB transmissions, with Fig. 7a showing the format of a 16-byte header of a transfer-based transfer descriptor for one endpoint, in accordance with the protocol, and Figs. 7b and 7c describing the contents of the header fields. The transfer-based protocol header consists of parameters that have the same definition as the conventional USB EHCI software, allowing the transfer descriptors to be easily constructed.

The transfer-based protocol also ensures that data can be sent to each USB endpoint on a fair basis.

Fig. 8 shows a situation in which the payload data associated with a first transfer descriptor TD1 is divided into three packets, PL1, PL2 and PL3, each of 64 bytes; the payload data associated with a second transfer descriptor TD2 comprises just one packet PL1 of 32 bytes; the payload data associated with a third transfer descriptor TD3 is divided into two packets PL1 and PL2, each of 8 bytes; and the payload data associated with a fourth transfer descriptor TD4 is divided into four packets PL1, PL2, PL3 and PL4, each of 16 bytes.

Fig. 9 illustrates the method by which these packets of data are transferred out of the RAM 50, or 56, to their respective endpoints in respective devices connected to the host.

As indicated by the arrow 90 in Fig. 8, a cyclical process occurs. Firstly, in step 91, the first packet PL1 associated with the first transfer descriptor TD1 is transferred. The transfer descriptor contains an Active flag which is set high, to indicate that there remains more data associated with this transfer descriptor.

Secondly, in step 92, the first packet PL1 associated with the second transfer descriptor TD2 is transferred. This transfer descriptor now contains an Active flag which is set low by the host controller 40, indicating that this completes the transfer of the payload data associated with the second transfer descriptor TD2.

Next, in steps 93 and 94, the first packets PL1 of payload data associated with the third and fourth transfer descriptors TD3 and TD4 respectively, are transferred. Again,

each of these transfer descriptors contain an Active flag which is set high, indicating that there is more of the payload data associated with each of the transfer descriptors, remaining to be transferred.

Next, in step 95, the second packet PL2 of payload data associated with the first transfer descriptor TD1 is transferred. The Active flag remains high, because there is still more of the payload data associated with that transfer descriptor, remaining to be transferred.

The transfer of the payload data associated with the second transfer descriptor TD2 has been completed, and so, in step 96, the second packet PL2 of payload data associated with the third transfer descriptor TD3 is transferred. This time, the Active flag in this transfer descriptor is set low, indicating that this completes the transfer of the payload data associated with the third transfer descriptor TD3.

In step 97, the second packet PL2 of payload data associated with the fourth transfer descriptor TD4 is transferred, and the Active flag remains high.

In step 98, the third packet PL3 of payload data associated with the first transfer descriptor TD1 is transferred, and the Active flag is set low, indicating that this completes the transfer of payload data associated with the first transfer descriptor.

In steps 99 and 100, the third and fourth packets PL3 and PL4 of payload data associated with the fourth transfer descriptor TD4 are transmitted, with the Active flag being set low in step 100, to indicate that this completes the transfer of the payload data associated with the fourth transfer descriptor TD4.

During execution of the transfer-based transfer descriptors, the content of the transfer-based transfer descriptors is updated by the host controller logic unit 52. For example, the Active flag within a transfer descriptor header is set low when the transfer of the payload data associated with the transfer descriptor is completed. The USB EHCI interface software 84 then reformats the updated transfer-based transfer descriptors into a format which can be handled by the conventional EHCI host stack 82, and the updated transfer-based transfer descriptors are copied back to the system memory 30.

There is therefore provided a host controller which allows the incorporation of high speed USB host functionality, in particular into non-PC based systems.